Using cursors in stored procedures

The MySQL code below is from a stackoverflow answer. It provides a design pattern with detailed explanation of the behavior of a cursor inside the stored procedure.

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `my_proc` $$
CREATE PROCEDURE `my_proc`(arg1 INT) -- 1 input argument; you might not need one
BEGIN

-- from http://stackoverflow.com/questions/35858541/call-a-stored-procedure-from-the-declare-statement-when-using-cursors-in-mysql

-- declare the program variables where we'll hold the values we're sending into the procedure;
-- declare as many of them as there are input arguments to the second procedure,
-- with appropriate data types.

DECLARE val1 INT DEFAULT NULL;
DECLARE val2 INT DEFAULT NULL;

-- we need a boolean variable to tell us when the cursor is out of data

DECLARE done TINYINT DEFAULT FALSE;

-- declare a cursor to select the desired columns from the desired source table1
-- the input argument (which you might or might not need) is used in this example for row selection

DECLARE cursor1 -- cursor1 is an arbitrary label, an identifier for the cursor
 CURSOR FOR
 SELECT t1.c1,
        t1.c2
   FROM table1 t1
```

```
  WHERE c3 = arg1;

-- this fancy spacing is of course not required; all of this could go on the same line.

-- a cursor that runs out of data throws an exception; we need to catch this.
-- when the NOT FOUND condition fires, "done" -- which defaults to FALSE -- will be set to
true,
-- and since this is a CONTINUE handler, execution continues with the next statement.

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

-- open the cursor

OPEN cursor1;

my_loop: -- loops have to have an arbitrary label; it's used to leave the loop
LOOP

  -- read the values from the next row that is available in the cursor

  FETCH NEXT FROM cursor1 INTO val1, val2;

  IF done THEN -- this will be true when we are out of rows to read, so we go to the statement
after END LOOP.
    LEAVE my_loop;
  ELSE -- val1 and val2 will be the next values from c1 and c2 in table t1,
       -- so now we call the procedure with them for this "row"
    CALL the_other_procedure(val1,val2);
    -- maybe do more stuff here
  END IF;
END LOOP;

-- execution continues here when LEAVE my_loop is encountered;
-- you might have more things you want to do here

END $$

DELIMITER ;
```

Source

Revision #2
Created 2023-02-08 12:22:53 UTC by bluecrow76
Updated 2023-02-08 12:30:46 UTC by bluecrow76